Interfejs graficzny aplikacji w Java

WSZYSTKIE PRZYKŁADY DO MATERIAŁU DOSTĘPNE POD ADRESEM: https://gitlab.com/examples4java/javafx oraz https://gitlab.com/examples4java/timelib

Obecnie większość aplikacji posiada swój interfejs graficzny. Niemal każdy użytkownik czuje się bardziej komfortowo gdy widzi np. wszystkie opcje danej aplikacji bądź są one dostępne przy kliknięciu myszą.

W obecnej chwili podczas projektowania interfejsu trzeba wziąć pod uwagę kilka czynników:
- powinien być przejrzysty
- opcje najczęściej użytkowane powinny być łatwo dostępne (najlepiej stale widoczne)
- interfejs powinien dostosowywać się do rozmiaru ekranu i/lub rozmiaru okna aplikacji
- interfejs powinien być lekki, tj. być przyjemny dla oka oraz nie zajmować zasobów systemowych (rzadko osiągane)

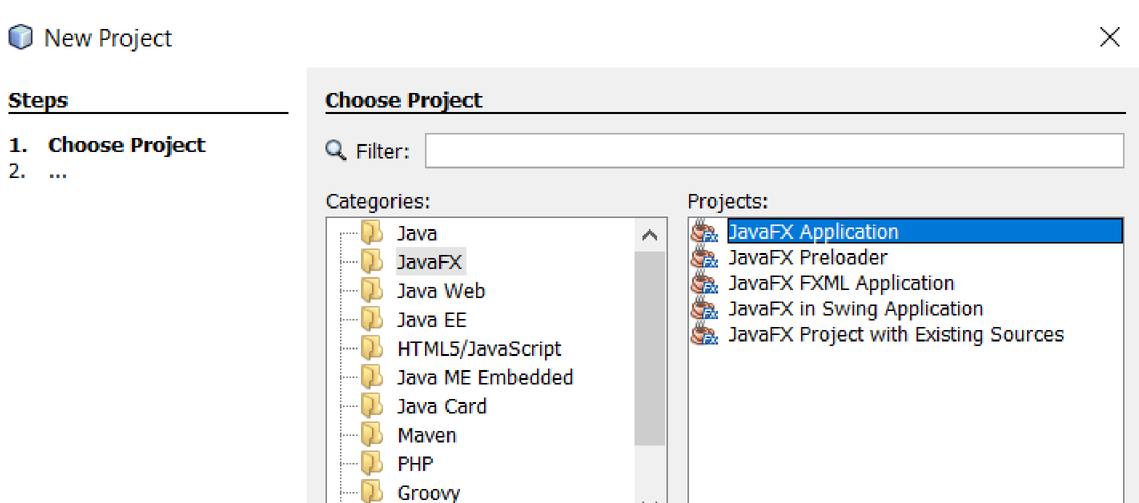
W przypadku Java dochodzi wymóg takiego samego wyglądu na wszystkich platformach, na których można uruchomić naszą aplikację, tj. wygląd powinien być niezależny od systemu bądź dostosowywać się do każdego systemu.

INFORMACJA: Coraz części jako interfejsu graficznego aplikacje wykorzystują dokumenty HTML bądź dokumenty budowane w oparciu o SGML/XML. W przypadku HTML często wykorzystywane jest okno, którego głównym elementem jest przestrzeń parsująca HTML (silnik przeglądarki internetowej). Najczęściej stosowanym silnikiem jest WebKit (darmowy, otwartoźródłowy silnik, stanowiący podstawę np. Safari, Opery, Chromium). Ten silnik dostępny jest też w Java.

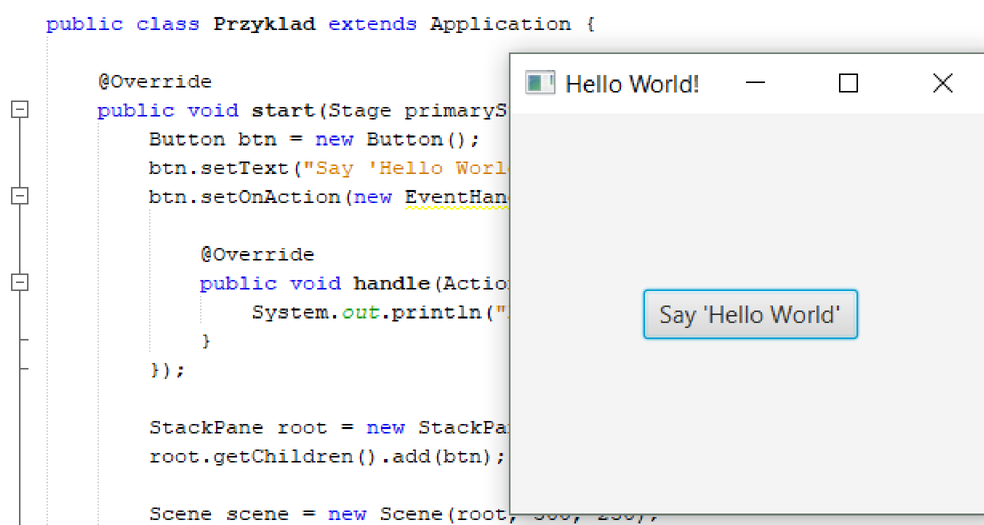
Java posiada kilka bibliotek umożliwiających tworzenie graficznych aplikacji:
- AWT – najstarsza implementacja okien; posiada najprostsze elementy okien, większość z nich trzeba wzbogacać o pożądaną funkcjonalność
- Swing – duchowy następca AWT; w rzeczywistości nadbudowuje bibliotekę o nowe kontrolki, a obecne wzbogaca o nową funkcjonalność. Obecnie nadal bardzo popularny
- JavaFX – zaprezentowany już w 2006 roku sposób tworzenia okien i wyglądu aplikacji w Java. JavaFX pozwala na tworzenie okien zarówno w sposób tradycyjny – bezpośrednio w kodzie, poprzez odpowiednie metody, jak i poprzez pliki z kodem JavaFX (JavaFX Script) wraz z możliwością nadawania im cech poprzez CSS. Rozwiązanie podziału na JavFX, CSS oraz klasę obsługującą wpisuje się w kanon modelu MVC (Model-View-Controller).

1. JavaFX

Aby utworzyć projekt JavaFX należy utworzyć dowolny projekt nowej aplikacji. W przypadku niektórych środowisk, jak np. NetBeans można wykorzystać predefiniowany projekt JavaFX:



Po utworzeniu projektu w ten sposób otrzymujemy prosty przykład aplikacji okienkowej, w której to po kliknięciu przycisku otrzymamy komunikat w konsoli.



Generalnie aplikacje w JavFX można tworzyć tym sposobem. W tym wypadku należy pamiętać o kilku rzeczach, opisanych w kolejnych punktach.

2. Przestrzeń okna.

W Java ustawianie elementów w oknie od zawsze wygląda nieco inaczej niż w jakiejkolwiek innej technologii. Każde okno ma odpowiednie przestrzenie, które wypełniamy elementami.

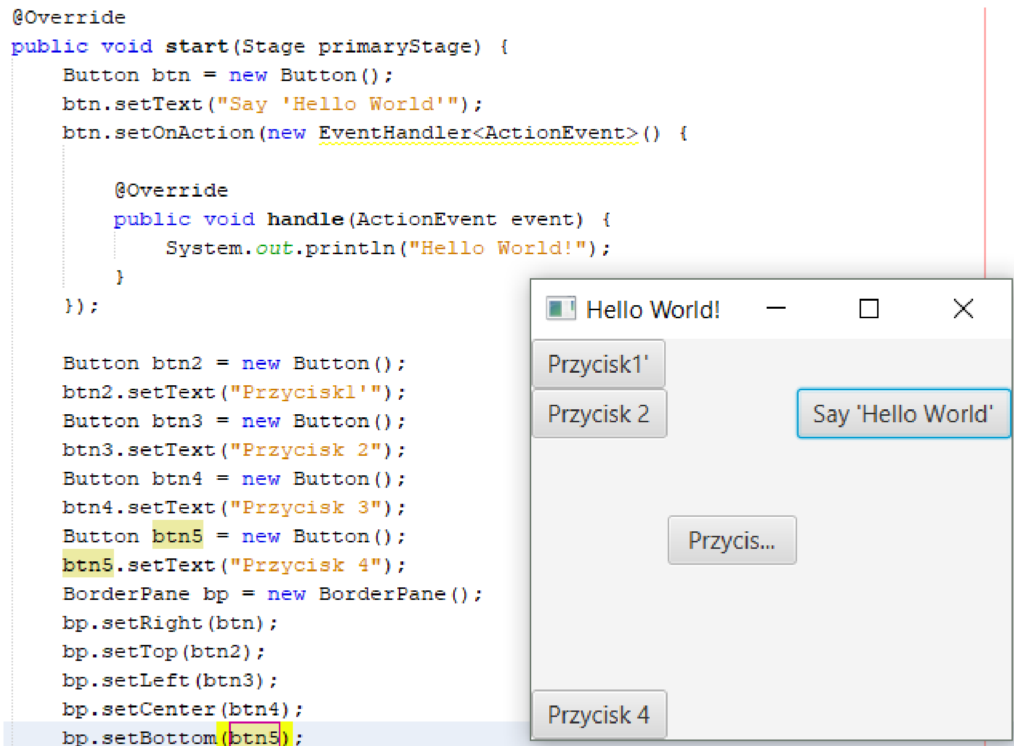
UWAGA: Prawdą jest, że np. w Swing można było pozycjonować bezwzględnie każdy element poprzez metodę setBounds(). Jednak niemal żaden projekt nie brał tego typu rozwiązania pod uwagę – wprowadzało to zbyt duży chaos do projektu (wedle programistów Java). Dlatego też tworzenie interfejsu w ten sposób jest bardzo niezalecany!

Ten typ zachowania jest w pełni odziedziczony po AWT oraz Swing. Dostępnymi przestrzeniami w JavaFX są:
a) BorderPane – ten wygląd jest najbardziej klasyczny. Podzielony jest na odpowiednie regiony:
- lewy
- prawy
- górny
- dolny

- środek

Każdy z nich może zawierać zarówno kontrolki jak i kolejne szablony (omówione w kolejnych podpunktach).

Poniżej zrzut przykładowego kodu oraz wygląd okna ustawiającego 5 przycisków na lewo, prawo, górę, środek oraz dół:

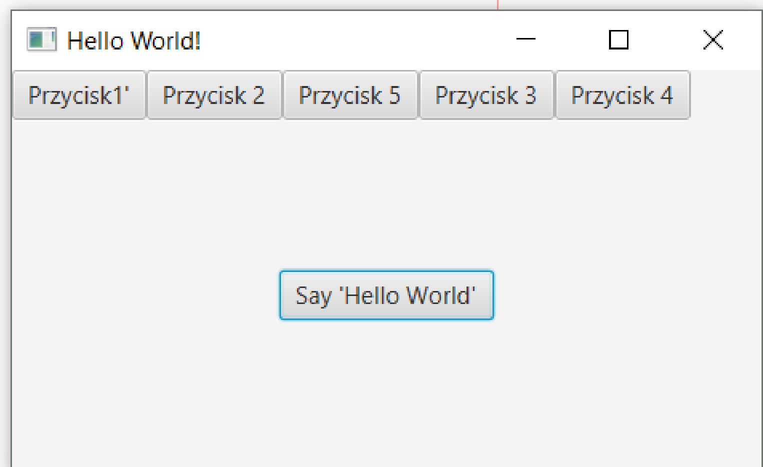


Jak można zauważyć, środkowy przycisk dostosowuje się do wielkości okna (gdyby je poszerzyć napis w nim byłby kompletny). To cecha BorderPane – wszystkie regiony domyślnie przyjmują wielkość największego elementu, środkowy zaś dostosowuje się do nich. Ponadto góra oraz dół zajmują przestrzeń od lewa do prawa, z pominięciem lewej i prawej części BorderPane (są one ograniczone od góry i dołu).

INFORMACJA: Jeżeli dodalibyśmy więcej niż jeden przycisk do danej przestrzeni to wyświetliłby się tylko ostatni. Dzieje się tak ponieważ poszczególne ramki zachowują się jak tzw. StackPane (czyli przestrzeń stosu). Każdy element jest układany jeden na drugim, przysłaniając poprzedni.

b) HBox – układ, który układa każdą nową kontrolkę jedną obok drugiej w poziomie. Pozwala to na ułożenie kilku elementów w ramach np. BorderPane w jednej z ram. Przykład wykorzystania (modyfikacja poprzedniego przykładu):

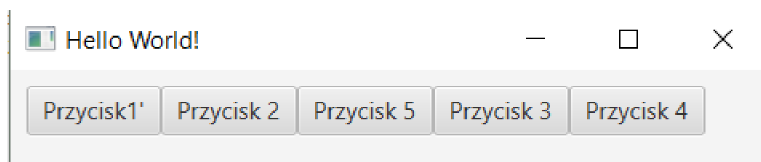
```java
btn.setText("Say 'Hello World'");
btn.setOnAction(new EventHandler<ActionEvent>() {

    @Override
    public void handle(ActionEvent event) {
        System.out.println("Hello World!");
    }
});

Button btn2 = new Button();
btn2.setText("Przycisk1'");
Button btn3 = new Button();
btn3.setText("Przycisk 2");
Button btn4 = new Button();
btn4.setText("Przycisk 3");
Button btn5 = new Button();
btn5.setText("Przycisk 4");
Button btn6 = new Button();
btn6.setText("Przycisk 5");
BorderPane bp = new BorderPane();

bp.setCenter(btn);

HBox hb = new HBox();
hb.getChildren().add(btn2);
hb.getChildren().add(btn3);
hb.getChildren().addAll(btn4,btn5);
hb.getChildren().add(2,btn6);
bp.setTop(hb);
```



Jak można zauważyć elementy do przestrzeni poziomej można dodawać po wywołaniu metody getChildren(), która po prostu zwraca kolekcję wszystkich elementów-dzieci tejże przestrzeni. Na przykładnie powyżej podane są 3 metody dodawania elementów:
- poprzez metodę add(), która dodaje nowy element na koniec kolekcji (wyświetlany jako ostatni)
- poprzez addAll() pozwalający podać całą kolekcję (w tym wypadku kolekcja tworzona jest dynamicznie poprzez dodanie kolejnych po sobie elementów)
- poprzez przeładowaną metodę add(), pozwalającą na podanie numeru indeksu, pod którym ma się znajdować nowy element (w przykładzie powyżej to indeks 2 – trzeci element)

Jak widać elementy są po sobie ustawione jeden po drugim.

HBox posiada dodatkowo następujące metody (najpopularniejsze):
- setPadding(Insets inset) - pozwala na ustawienie odbić od krawędzi elementu nadrzędnego; można ustawiać zarówno odbicie od każdej krawędzi indywidualnie (lewo, góra, prawo, dół) jak i wszystkie jednakowo. Jako parametr setPadding przyjmuje element Insets. Przykład:
hb.setPadding(new Insets(10)); - ustawia odbicie na 10 pikseli (zrzut poniżej)
hb.setPadding(new Insets(0,5,10,5)); - ustawia odbicie od prawej na 5, od dołu na 10 oraz od lewej na 5 (góra ustawiona na zero)



- setSpacing(double spacing) - pozwala na ustawienie odstępów (w pikselach) pomiędzy wyświetlanymi elementami
- setOpacity(double opacity) – pozwala na ustawienie nieprzezroczystości elementu (np. 0.5 będzie odpowiednikiem 50% nieprzezroczystości elementu). Trzeba pamiętać, że wszystkie dzieci elementu też staną się nieprzezroczyste (naturalne zachowanie)

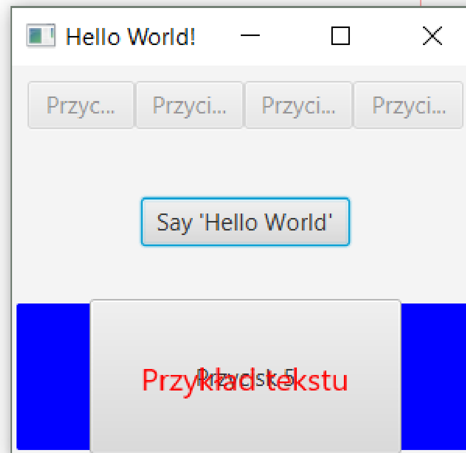
c) VBox – układ analogiczny do poprzedniego, pozwalający na ustawienie elementów pionowo względem siebie. Posiada te same możliwości co poprzednik

d) StackPane – element domyślnie dostępny w przykładowym programie. Stosowany jako domyślny układ w niemal każdym układzie okna. Jego właściwość można wykorzystać do budowania elementów wielopoziomowych, nadających specyficzny wygląd elementu (np. ramka wokół przycisku) lub utworzenie wyglądu aplikacji jako slajdu (po kliknięciu pojawiają się dodatkowe opcje spod aktualnego elementu).

Ponieważ jest to stos, działa jak stos – ostatnie na wejściu, pierwsze na wyjściu.

Przykład utworzenia przyciski z dodatkowym elementem tła:

```
Button btn6 = new Button();
btn6.setPrefSize(200, 100);
btn6.setText("Przycisk 5");
Text txt = new Text("Przykład tekstu");
txt.setFont(new Font(STYLESHEET_MODENA, 20));
txt.setFill(Color.RED);
Background ground = new Background(new BackgroundFill(Color.BLUE, new CornerRadii(2), new Insets(3)));
StackPane sp = new StackPane();
sp.setBackground(ground);
sp.getChildren().addAll(btn6,txt);
BorderPane bp = new BorderPane();

bp.setCenter(btn);

HBox hb = new HBox();
hb.setOpacity(0.5);
hb.setPadding(new Insets(10));
hb.getChildren().add(btn2);
hb.getChildren().add(btn3);
hb.getChildren().addAll(btn4,btn5);
//hb.getChildren().add(2,btn6);
bp.setTop(hb);

VBox vb = new VBox();
vb.getChildren().add(sp);
bp.setBottom(vb);
```

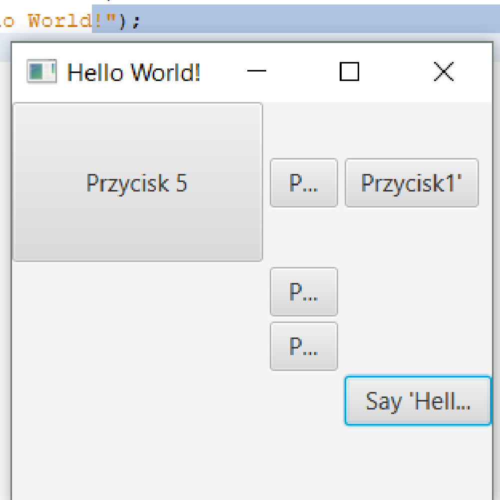


Jak widać, przycisk zyskał inną wielkość (metoda setPrefSize()). Dodatkowo pole StackPane (tutaj reprezentowane jako sp) zyskało tło (klasa Background), które ma zaokrąglone rogi (2 piksele) oraz jest oddalone o 3 piksele od granic elementu nadrzędnego (w tym wypadku sp). Utworzony został jeszcze tekst, któremu nadano 20 pikseli wielkości oraz kolor czerwony. Na uwagę zasługuje tutaj dodawanie elementów do StackPane – tło jest osobnym bytem (setBackground(Background bg)), natomiast przycisk i tekst dodawane są tą samą metodą co w przypadku dodawania elementów do HBox/VBox. Gdyby dodać elementy w odwrotnej kolejności, wtedy przycisk przysłonił by napis.

Na koniec sp zostało dodane do dolnej części utworzonego wcześniej BorderBox.

e) GridPane – ten układ jest jednym z częściej stosowanych. Pozwala on rozłożyć wszystkie elementy jak w tabeli (wierszami i kolumnami). Poniżej przedstawiony przykład pokazuje ułożenie elementów:

```java
public void start(Stage primaryStage) {
    GridPane gpane = new GridPane();
    gpane.setHgap(4);
    gpane.setVgap(3);

    Button btn = new Button();
    btn.setText("Say 'Hello World'");
    btn.setOnAction(new EventHandler<ActionEvent>() {

        @Override
        public void handle(ActionEvent event) {
            System.out.println("Hello World!");
        }
    });

    Button btn2 = new Button();
    btn2.setText("Przycisk1'");
    Button btn3 = new Button();
    btn3.setText("Przycisk 2");
    Button btn4 = new Button();
    btn4.setText("Przycisk 3");
    Button btn5 = new Button();
    btn5.setText("Przycisk 4");
    Button btn6 = new Button();
    btn6.setPrefSize(200, 100);
    btn6.setText("Przycisk 5");

    gpane.add(btn, 2, 3);
    gpane.addRow(0, btn6,btn5,btn2);
    gpane.addColumn(1, btn4,btn3);

    Scene scene = new Scene(gpane, 300, 250);
```



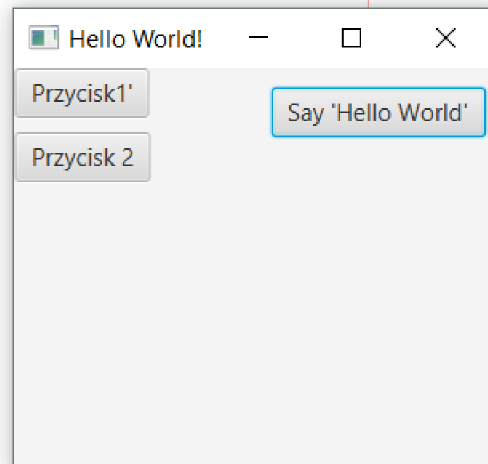
Podstawowymi metodami są tutaj setHgap(int) oraz setVgap(int). Pozwalają one na utworzenie odpowiedniej liczny wierszy oraz kolumn.

Dodawanie do siatki wygląda podobnie jak w przypadku pozostałych układów z tą różnicą, że element trzeba odpowiednio umiejscowić:
- add(Node element, int column, int row) – umiejscawia element w odpowiedniej komórce
- addRow(int rowNum, Node… elements) – dodaje do wskazanego wiersza nowe elementy
- addColumn(int columnNum, Node… elements) – analogicznie do wierszy dodaje kolumny

Dodawanie siatki jest analogiczne do dodawania poprzednich układów (dodajemy go jako element sceny okna).

f) AnchorPane – wygląd jest tutaj ustawiany względem zakotwiczenia elementów względem granic tego elementu. Można zatem stwierdzić, że jest podobny do stosowanego BorderPane z tą jednak różnicą, że programista sam może wyznaczyć w jakiej relacji do danej krawędzi będzie dodawany element (np. ile pikseli ma być od niej przesunięty).

```java
public void start(Stage primaryStage) {

    AnchorPane ap = new AnchorPane();
    Button btn = new Button();
    btn.setText("Say 'Hello World'");
    btn.setOnAction(new EventHandler<ActionEvent>() {

        @Override
        public void handle(ActionEvent event) {
            System.out.println("Hello World!");
        }
    });

    Button btn2 = new Button();
    btn2.setText("Przycisk1'");
    Button btn3 = new Button();
    btn3.setText("Przycisk 2");
    Button btn4 = new Button();
    btn4.setText("Przycisk 3");
    Button btn5 = new Button();
    btn5.setText("Przycisk 4");
    Button btn6 = new Button();
    btn6.setPrefSize(200, 100);
    btn6.setText("Przycisk 5");

    ap.getChildren().addAll(btn,btn2,btn3/*,btn4,btn5,btn6*/);
    ap.setTopAnchor(btn, 12D);
    ap.setRightAnchor(btn, 5D);
    ap.setLeftAnchor(btn2, 0.5D);
    AnchorPane.setTopAnchor(btn3, 40D);
    AnchorPane.setLeftAnchor(btn3, 0.5D);

    Scene scene = new Scene(ap, 300, 250);
```



Jak widać na powyższym zrzucie, elementy AnchorPane dodaje się analogicznie jak do poprzednich elementów. Największą zmianą jest możliwość ich ustawiania poprzez metody
setTopAnchor(Node element, Double val)
setBottomAnchor(Node element, Double val)
setLeftAnchor(Node element, Double val)
setRightAnchor(Node element, Double val)

Należy zauważyć, że na zrzucie powyżej metody te użyte są zarówno w odniesieniu do naszego obiektu, jak i bezpośrednio z klasy; są to metody statyczne, które bezpośrednio wpływają na cały element AnchorPane, po wskazaniu odpowiedniego elementu wewnątrz AnchorPane (należy zauważyć, że wszystkie elementy dodawane muszą być unikatowe, nie ma więc możliwości pomyłki w przesuwaniu nawet elementów należących do różnych AnchorPane)

INFORMACJA: Zalecane jest używanie metody poprzez klasę (np. AnchorPane.setTopAnchor()) zamiast poprzez obiekt!

Dodatkowymi elementami pozycjonującymi są jeszcze :
- FlowPane – element umieszczający elementy w ciągu poziomym lub pionowym. Element stanowi połączenie właściwości HBox/VBox.
- TilePane – element dodający kafle. Stanowi niejako rozszerzenie elementu GridPane – również pozwala na pozycjonowanie w komórkach, jednak każda z nich będzie miała równy przydział miejsca.

Program demonstracyjny nie zawiera pokazu działania tych elementów. Przykład dostępny w materiałach dodatkowych (pozycja 3).

3. Najważniejsze elementy (kontrolki) dostępne w JavaFX

a) Button – dodaje nowy przycisk; głównie pozwala na modyfikacje takie jak tekst, wartości tekstu (setFont()), tła oraz wielkości (szerokość i wysokość).
b) CheckBox – dodaje element, który można zaznaczyć bądź nie (pole wyboru); posiada podobne opcje do przycisku
c) ProgressBar – dodaje element umożliwiający pokazanie użytkwnikowi stan postępu naszej aplikacji (np. kopiowania plików)
d) Hyperlink – element pozwalający na dodawanie hiperłączy do okna; działają tak samo jak łącza na stronach WWW
e) ChoiceBox<T> - element dodający rozwijalna listę (często nazywany combobox lub select). Lista przechowuje elementy określonego typu; mogą to być inne kontrolki, czysty tekst bądź obiekty własne. W przeciwieństwie do innych elementów, dostęp do zawartości (listy) uzyskuje się poprzez metodę getItems().
f) ProgressIndicator – unowocześniony element ProgressBar; obecnie w modzie jest podawanie użytkownikowi informacji o procesie przetwarzania, nie zaś precyzyjnego (mniej lub bardziej) czasu zakończenia zadania (bądź procencie ukończenia).
g) ScrollBar – jeden z podstawowych elementów graficznych okna. Pozwala on na umieszczanie treści większych niż okno/wielkość elementu zawierającego treść wykraczającą poza element.
h) RadioButton – podobny do CheckBox, jednak pozwala jednak na wybór tylko jednego pola z całej grupy. Przyciski tego typu, by działały poprawnie, muszą być przydzielane do dodatkowej kontrolki – ToggleGroup.

Istnieją jeszcze inne kontrolki. Można je znaleźć np. na stronie z materiałów (4). Przykład działania RadioButton można znaleźć na stronie pod numerem 5.

4. Zdarzenia

Wszystkie operacje związane z kontrolkami są zarządzane przez zdarzenia. Zdarzenia można przyrównać do przerwań np. w mikrokontrolerach.

Generalnie komputery, w tym mikroprocesory, pracują wykonując zadanie. Zadanie są czysto teoretycznie nie do przerwania. Jednak, dzięki implementacji przerwań, stan tej rzeczy się zmienia. Przykładowym przerwaniem może być np. podanie sygnału wysokiego na jedną z nóżek procesora. Wtedy to, w zależności od zdefiniowanej przez projektanta obsługi tego zdarzenia, procesor przerywa pracę i np. wyświetla tekst na ekranie (nie w sensie stricte on, lecz to on podejmuje odpowiednie procedury i wywołuje inne, dodatkowe przerwania celem zmiany koloru/naświetlenia matrycy, która wyświetli nam dane).

Wracając do komputera, użytkownik ma do dyspozycji następujące możliwości wywołania przerwań:
- klawiatura – poprzez nacisk, przytrzymanie bądź zwolnienie przycisku, serii przycisków bądź kombinacji przycisków
- mysz – poprzez poruszenie myszą, poprzez najechanie kursorem na wskazany element, wciśnięcie przycisku myszy (obecnie myszy mogą mieć ich dowolną ilość aczkolwiek rozważa się lewy/prawy przycisk oraz środkowy – suwak), przesuwanie kółka (szczególnie wykorzystywane przy przybliżaniu i oddalaniu)
- tablet graficzny – poprzez nacisk na matrycę odpowiednim rodzajem końcówki pióra, poprzez nacisk na matrycę palcem, poprzez wskazanie nad matrycą, poprzez siłę nacisku
- skaner – poprzez wprowadzenie obrazu/obiektu metodą czytania elementów matrycą światłoczułą
- mikrofon – wprowadzanie dźwięku

Oczywiście istnieje jeszcze kilka innych metod wywołania przerwań przez użytkownika. W obecnych czasach nawet drukarka ma możliwość wygenerowania przerwania (brak papieru czy tuszy). W tym momencie dochodzimy do miejsca, gdzie dowiadujemy się, iż przerwania mogą być generowane także przez urządzenia. Ściśle rzecz ujmując, także przez sam komputer (w tym procesor). Tym samym przerwania dzielą się na dwie grupy:

- foreground (podkładowe/zwierzchnie) – te są generowane najczęściej przez użytkownika
- background (spodnie) – te są generowane przez sprzęt i/lub sam sprzęt komputerowy

Większość przerwań pierwszego typy można obsłużyć poprzez odpowiednią klasę EventHandler. Każdy z obiektów posiada odpowiednią metodę pozwalającą związać działanie wskazanego obiektu np. na kliknięcie myszą z odpowiednią metodą wykonywaną gdy warunek zostanie spełniony (np. kliknięcie na przycisk spowoduje zamknięcie programu).

Najpopularniejsze zdarzenia, dostępne dla większości kontrolek w JavaFX:
- onMouseClickedProperty() - zdarzenie generowane jest w przypadku kliknięcia myszą
- onMousePressedProperty() - zdarzenie generowanie w chwili gdy przycisk myszy jest wciśnięty (przytrzymany)
- onMosueReleasedProperty() - zdarzenie generowane w chwili puszczenia przycisku myszy (był przytrzymywany i został puszczony)
- onMouseEnteredProperty() - kursor myszy jest w polu kontrolki
- onMouseExitedProperty() - kursor myszy opuścił pole kontrolki
- onMouseMovedProperty() - kursor myszy porusza się w polu kontrolki
- onKeyPressedProperty() - klawisz klawiatury został wciśnięty (i jest przytrzymywany)
- onKeyReleasedProperty() - klawisz klawiatury został zwolniony (był wcześniej przytrzymywany)
- onKeyTypedProperty() - klawisz został naciśnięty (odpowiednik kliknięcia myszą)

Każdemu zdarzeniu można dopisać odpowiednią operację poprzez metodę set().

Przykładowo jeżeli chcemy przyciskowi przypisać operację do zdarzenia kliknięcia, możemy zrobić w następujący sposób:

```
Button btn = new Button();
        btn.setText("Say 'Hello World'");

        Label l = new Label();
        btn.onMouseClickedProperty().set(new EventHandler<MouseEvent>() {
            @Override
            public void handle(MouseEvent event) {
                if (event.getButton() == MouseButton.PRIMARY)
                    l.setText("Wciśnięto lewy przycisk");
                if (event.getButton() == MouseButton.SECONDARY)
                    l.setText("Wciśnięto prawy przycisk");
            }
        });

        btn.onMouseMovedProperty().set(new EventHandler<MouseEvent>() {
            @Override
            public void handle(MouseEvent event) {
                l.setText("Mysz znajduje się w pozycji: " + event.getSceneX() + ", " +
event.getSceneY());
            }
```

```
        });
```

Powyższy kod zawiera dwa elementy:
- przycisk (Button)
- etykietę (Label)

Jako układ wybrana została siatka (GridPane). Na początek tworzymy wszystkie kontrolki. Następnie dla przycisku przypisujemy dwa zdarzenia:

- kliknięcia myszą – zdarzenie to sprawdza, który przycisk myszy został kliknięty - PRIMARY to klawisz główny, SECONDARY to klawisz pomocniczy; w Java specjalnie zastosowano taką konwencję nazw – niektórzy mogą zmieniać układ klawiszy lewy na prawy. Chociaż nie miałoby to znaczenia dla kodu, jest to czytelniejsze dla programisty (czego naprawdę oczekuje)

Po sprawdzeniu zostaje wyświetlony odpowiedni komunikat na etykiecie.

- poruszania myszą – zdarzenie wywołuje się gdy użytkownik porusza kursorem nad wskazanym elementem.

Oczywiście można również tworzyć obiekty zdarzeń przypisywane do zmiennych. Wtedy możliwe jest przypisanie tej samej akcji do kilku różnych obiektów (operacje mogą być powielane). Wszystko zależy od aktualnych potrzeb. (więcej w materiałach 9 oraz 10).

Przykład kodu znajduje się w repozytorium.

5. Przypisywanie stylów do kontrolek.

W JavaFX możliwe jest stylizowanie elementów. Style odpowiadają za takie cechy jak:
- font
- kolor tła
- kolor elementu
- obramowanie
- zmianę wyglądu na określone zdarzenia

Style można dodawać bezpośrednio w kodzie aplikacji. Przykładowo jeżeli chcemy nadać odpowiedni wygląd dla naszego przycisku, możemy zrobić to tak:

```
 Button btn = new Button();
        btn.setText("Say 'Hello World'");
        btn.setStyle("-fx-background-color: #00FF00; -fx-text-fill: #FFFFFF; ");
```

Powyższe linie utworzą nam przycisk, któremu nadadzą styl:
- tło przycisku (jego powierzchnia) będzie miała kolor zielony (zapisany szesnastkowo, można użyć nazwy green i jej pochodnych, np. darkgreen)
- kolor tekstu będzie miał kolor biały (szesnastkowo maksymalne wartości, można użyć nazwy white)

W przypadku zmiany wartości np. przy najechaniu myszy można posłużyć się tzw. pseudoklasami stylu. Niestety w wypadku operowania kodem są one nieosiągalne. Można natomiast nadać pożądane wartości CSS poprzez zdarzenia:

```
btn.onMouseEnteredProperty().set(new EventHandler<MouseEvent>() {
        @Override
        public void handle(MouseEvent event) {
           btn.setStyle("-fx-background-color: #000000; -fx-text-fill: #FFFFFF; ");
        }
     });

     btn.onMouseExitedProperty().set(new EventHandler<MouseEvent>() {
        @Override
        public void handle(MouseEvent event) {
           btn.setStyle("-fx-background-color: #00FF00; -fx-text-fill: #FFFFFF; ");
        }
     });
```

Zdarzenie na najazd myszą zmieni nam kolor przycisku na czarny (kolor tekstu można było pominąć, gdyż jest taki sam). W przypadku odjazdu myszy zmieniamy wartość na pierwotną.

Powyższy kod można poprawić poprzez wstawienie stałych ciągów znakowych (final), dzięki czemu zmian wartości można poprawić późniejszą obsługę kodu (zmiana wartości CSS będzie odbywać się w jednym miejscu, a nie jak w powyższym przykładzie w 3 miejscach).

Najważniejsze selektory JavaFX (pełna lista dostępna w materiale 11):
a) ramka
- -fx-border-color – ustawienie koloru dla obramowania kontrolki
- -fx-border-width – ustawienie szerokości obramowania (wartości dodatnia)
- -fx-border-radius – ustawienie zaokrągleń ramek

przykład:
-fx-border-color: #0000FF;
-fx-border-width: 2;
-fx-border-radius: 5 5 5 5;

b) tło
- -fx-background-color – ustawienie koloru tła
- -fx-background-image – ustawienie obrazu jako tła (podawany w ścieżce)
- -fx-background-radius – ustawienie zaokrągleń tła (niezależne od ramek)
- -fx-background-insets – ustawienie wartości okna (regionu), w którym wyświetlane będzie tło lub obrazek tła. Podawane czwórkami rozpoczyna się od góry (góra, prawo, dół, lewo). Można podać jedną wartość (wtedy wszystkie regiony przyjmą taką samą). Tak więc element ten powstaje poprzez odsunięcie go od krawędzi elementu nadrzędnego (w tym wypadku obramowania elementu)

Przykład
-fx-background-color: #00FF00;
-fx-background-image: './obrazy/obraz.jpg';
-fx-background-radius: 5;
-fx-background-insets: 10;

c) tekst
- -fx-font-family – ustawienie konkretnego fontu bądź rodziny fontów dla elementu
- -fx-font-size – ustawienie wielkości czcionek fontu
- -fx-font-style – pozwala na ustawienie np. pochylenia tekstu

- -fx-font-weight – pozwala ustawić grubość tekstu

d) różne
- -fx-alignment – wyrównanie wskazanego obiektu bądź elementu obiektu do jednej z predefiniowanych wartości (np. left, left-center, top, bottom, bottom-right itd.)
- -fx-spacing – tworzenie odstępów od obiektu (pomiędzy innymi obiektami)

6. Tworzenie interfejsu w oddzielnych plikach.

Najnowszym trendem w programowaniu aplikacji z interfejsem jest zupełne oddzielenie kodu interfejsu, kodu programu oraz logiki połączenia pomiędzy interfejsem a kodem programu. Wynika to ze specjalizacji programistów:
- tworzących logikę biznesową, nazywaną często wewnętrzną, spodnią lub z angielskiego back-end (wykończenie tylne)
- tworzących wygląd strony, stany przejść pomiędzy opcjami czy też rozkładem opcji na poszczególnych ekranach czyli tzw. projekt wyglądu, część wierzchnia bądź z angielskiego design/front-end (wykończenie przednie)

Pierwszą z tych części nazywa się modelem. Odpowiada ona za pobieranie informacji z baz danych, przetwarzaniem informacji zebranych od użytkownika w toku użytkowania aplikacji, komunikację poprzez sieć, generowanie informacji zwrotnych dla użytkownika itd.

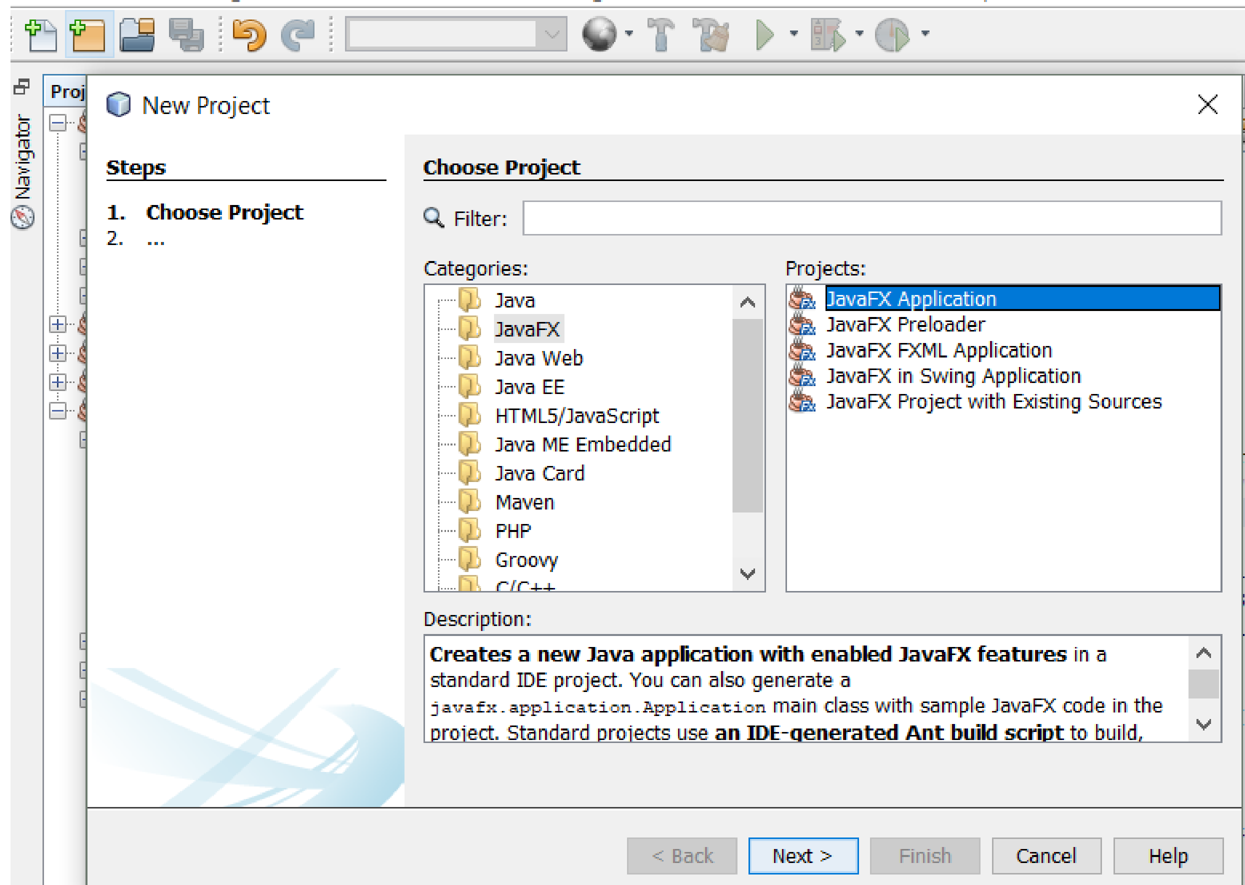
Druga część nazywana jest widokiem. Widok zawiera animacje interfejsu, zmiany wyglądu powodowane działaniami użytkownika, zmianę układu w określonej sytuacji (opcje programu, zmiana rozdzielczości ekranu, zmiana ekranu itp.), czy też dokładanie bądź usuwanie dodatkowych elementów interfejsu (np. klawiatura ekranowa w zależności od typu ekranu).

Dodatkowo aplikacje posiadają często część kodu odpowiedzialną tylko za połączenie funkcji programu z interfejsem użytkowym, przykładowo naciśnięcia przycisku bądź ruchu myszy nad elementem z daną funkcją programu. Część tą nazywa się kontrolerem, z racji pełnionej funkcji. Chociaż pisana przez programistów, często funkcje do elementów generowane są automatycznie, jak przykładowo przyciśnięcie przycisku Ok (wygenerowana nazwa może brzmieć onOkButtonClick) czy Wykonaj (onWykonajButtonClick). Nazwy te biorą się z nazw kontrolek, do których są dołączone (jak można mniemać, pierwsza z nich nazywa się okButton, druga wykonajButton), z nazwy zdarzenia, na które ma nastąpić reakcja (tutaj po prostu kliknięcie), natomiast przedimek wskazuje, iż funkcja wywoła się w reakcji na coś (jest zdarzeniem). Trzeba brać pod uwagę fakt, że nie zawsze nazwy funkcji muszą być dokładnie takie – wszystko zależy czy są faktycznie generowane (mogą być napisane ręcznie), w jaki sposób generuje je środowisko/narzędzie programistyczne oraz czy projekt nie zakłada innego nazewnictwa (np. lokalizującego nazwy w kodzie – chociaż to zdarza się niezwykle rzadko).

Model programistyczny, który zezwala na taki podział projektu (model, widok oraz kontroler) nazywa się MVC (z angielskiego Model, View, Controller). Obecnie jest jednym z częściej stosowanych w programowaniu obiektowym (i nie tylko).

Technologia JavaFX generalnie była projektowana pod właśnie układ MVC. Chociaż przedstawione przykłady wkładają kod JavaFX bezpośrednio do kodu aplikacji, możliwe jest przechowanie go w osobnych plikach. Struktura tych plików bazuje na XML (eXtensible Markup Language). W Java pliki te posiadają rozszerzenie fxml (FX XML). Dodatkowo możliwe jest dodawanie plików CSS (Cascade Style Sheets), plików znanych wszystkim projektantom stron WWW. Tworzenie formatek (okien) przypomina zatem tworzenie strony HTML. Pliki można tworzyć samodzielnie bądź, przykładowo używając NetBeans, wykorzystać wygodny automat.

Dodanie interfejsu graficznego może nastąpić w dowolnym momencie tworzenia aplikacji. Można też od razu tworzyć projekt graficzny (z przykładowym kodem). W tym celu wybieramy New Project, a w kategoriach zaznaczamy JavFX.



W tym momencie można utworzyć jeden z gotowych, predefiniowanych projektów:
a) JavaFX Application tworzy nam nowy projekt z przykładową sceną StackPane, na której będzie jeden przycisk.
b) JavaFX Preloader utworzy nam podobną aplikację, jednak z ukazaniem zastosowania JavaFX w przypadku okienka ładowania (często nazywanego splash screen). Ma ono na celu „zajęcie" użytkownika na czas ładowania właściwej aplikacji (tego typu opcja często wykorzystywana jest np. podczas ładowania Adobe Photoshop czy Mircosoft Word)
c) JavFX FXML Application to z kolei rozwiązanie, które powinno nas zainteresować na tym etapie nauki. Tworzy bowiem projekt z przykładowym plikiem fxml oraz css, a także kontrolerem (klasa w Java odpowiedzialna na połączenie logiki interfejsu i wykonania).
d) JavaFX in Swing Application pozwala na połączenie nowej technologii ze starymi nawykami programistycznymi. Generalnie nie jest to dobre rozwiązanie dla nowej aplikacji (jedynie dla istniejącej już aplikacji, którą chcemy przebudować w miarę bezbolesny dla nas sposób)
e) JavaFX Project with Existing Sources łączy nowy projekt aplikacji JavaFX z istniejącymi już klasami aplikacji klasycznej (konsolowej).

6.1 Tło programu, szkielet z konsoli

Ponieważ utworzenie nowego projektu wedle schematu jest bardzo proste przyjmiemy rozwiązanie, że posiadamy już nasz projekt, do którego chcemy dołożyć interfejs.

Obecnie posiadamy program o nazwie StopWatch. Program pozwala na mierzenie czasu, zapisania zmierzonego czasu, a także przejrzenia zapisanych czasów (po zapisanej nazwie). W celu mierzenia

czasu wykorzystamy wcześniej stworzoną bibliotekę do mierzenia czasu (dołączamy ją do projektu).
Sam program wykorzystuje do czytania danych, zamiast klasy Scanner, podstawowe mechanizmy strumienia wejściowego. Rozwiązanie to ma na celu pokazanie sposobów alternatywnych na pozyskanie informacji od użytkownika.

Ważnym aspektem w programie konsoli było pozyskanie odpowiedniego kodowania znaków. Domyślnie JVM pracuje z kodowaniem UTF-8. Niestety system Windows domyślnie nie korzysta z tego kodowania, lecz ze swojego własnego (dla każdej grupy językowej innego, kodowanie dla języka polskiego to Windows-1250). Z tego też powodu konsola w Java może, zamiast liter ą, ć, ę, ł, ń, ó, ś, ż, ź wyświetlić, w najlepszym wypadku, kwadraty.

Ponieważ nie możemy użytkownikowi zabronić korzystania z jego liter narodowych, problem ten musi zostać rozwiązany. Po pierwsze musimy się upewnić, w jakim kodowaniu działa JVM. Od wersji 7 można wykorzystać wbudowany mechanizm sprawdzania kodowania znaków:

new InputStreamReader(System.in).getEncoding() - metoda zwróci wartość tekstową nazwy kodowania używaną przez maszynę Java (w większości przypadków będzie to UTF-8) podczas wpisywania znaków do konsoli.

new OutputStreamWriter(System.out).getEncoding() - metoda zwróci wartość tekstową nazwy kodowania używaną przez maszynę Java w przypadku wyświetlania tekstu w konsoli.

Jeżeli chcemy wymusić na naszym programie by wyświetlał znaki w określonym kodowaniu możemy wstawić, na początku programu, taki kod:

```
try {
    System.setOut(new PrintStream(System.out, true, "utf-8"));
 } catch (UnsupportedEncodingException ex) {
    Logger.getLogger(StopWatch.class.getName()).log(Level.SEVERE, null, ex);
 }
```

pogrubiona linia odpowiada za nadanie systemowemu strumieniowi wyjścia odpowiedniego kodowania (w tym wypadku ręcznie wpisana wartość utf-8). Wartość można pozyskać również poprzez którąś z wyżej podanych metod.

INFORMACJA: Najważniejszym punktem jest kodowanie WYŚWIETLANIA ZNAKÓW, nie zaś wprowadzania. Wprowadzanie jest „obojętne” ponieważ zawiera surowe wartości bajtów. To określenie wyświetlenia nada odpowiednie wartości wyświetlanych znaków!

Niestety samo nadanie wartości w kodzie niekoniecznie przyniesie zamierzony efekt w samym programie. Dodatkowo musimy wymusić kodowanie w samym kompilatorze/procesorze Java. W przypadku NetBeans należy zastosować się do uwagi zgłoszonej w materiale 12 – trzeba w pliku netbeans.conf (katalog etc w ścieżce instalacyjnej środowiska) dodać w linii

netbeans_default_options

dodatkową opcję w postaci

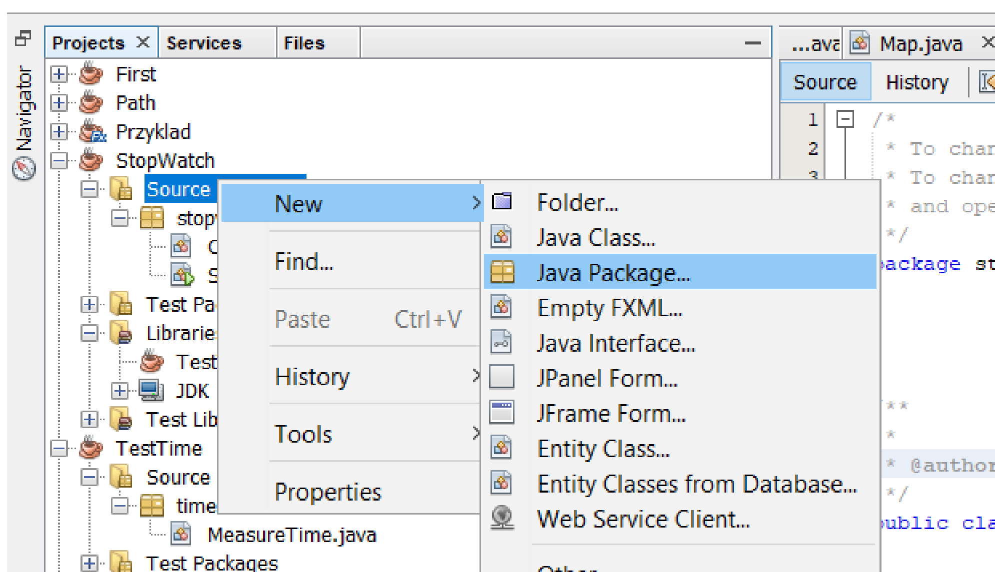
-J-Dfile.encoding=UTF-8

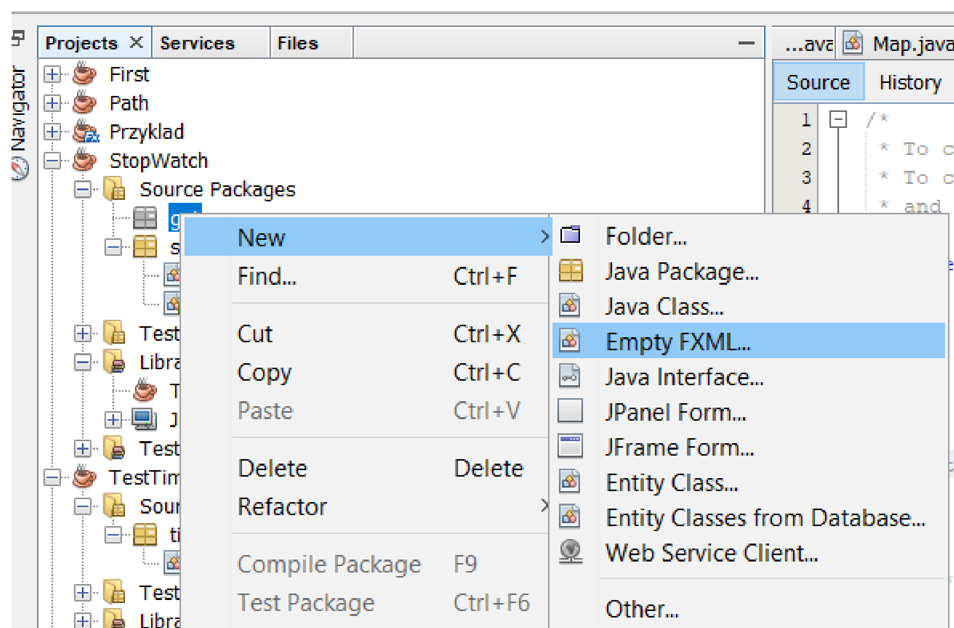
(opcję dodaje się do istniejących – jako separator kolejnych opcji służy spacja)

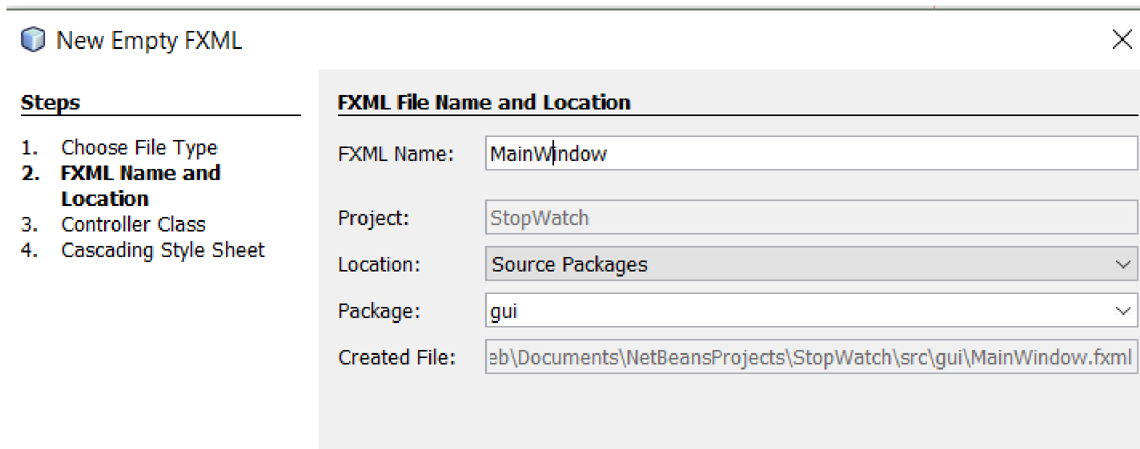
Reszta omawianej aplikacji jest podobna do pozostałych, tworzonych na zajęciach. Ewentualne nowości i niejasności wyjaśnione są w komentarzach do kodu. Aplikacja jest w pełni funkcjonalna, można jej bez przeszkód używać.

Podczas jej użytkowania można jednak zauważyć pewne trudności. Każda opcja musi być wyświetlana odpowiednim poleceniem. Każdorazowo musimy wykonywać operacje sekwencyjnie. Znacznie prościej byłoby obsługiwać ten prosty programik poprzez interfejs użytkownika.

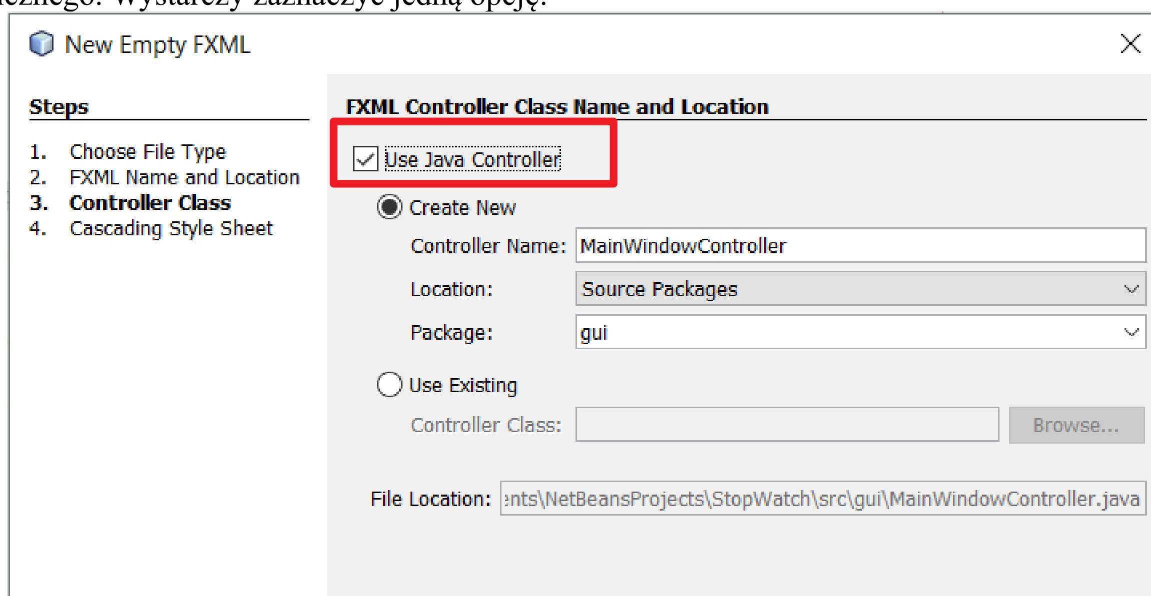
Na początek stwórzmy nową paczkę o nazwie gui celem nie zaśmiecania obecnego kodu. W paczce dodamy nowy plik fxml:



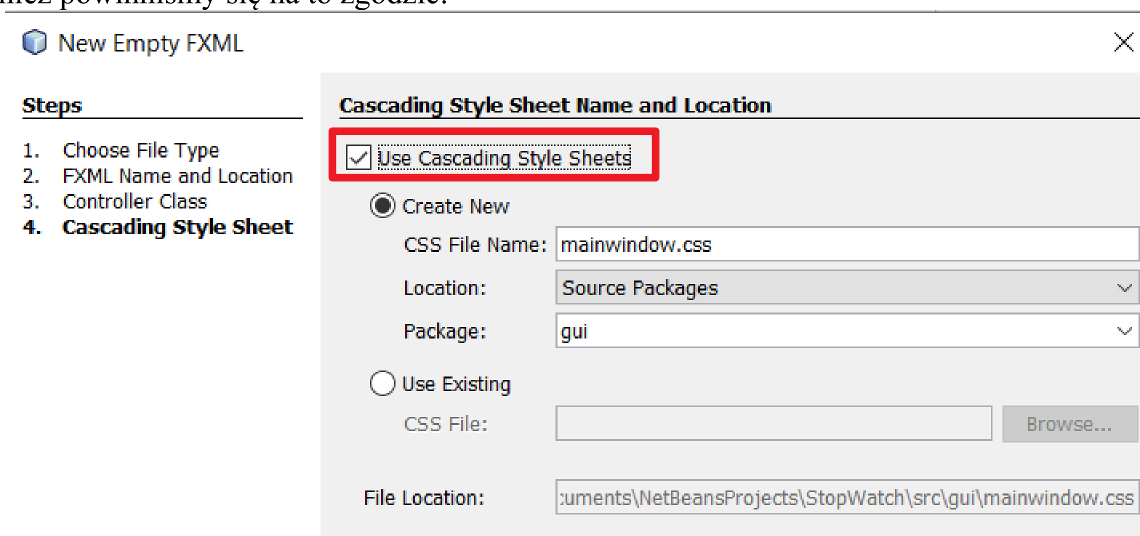




Dobrym rozwiązaniem jest użycie kreatora do utworzenia pliku klasy kontrolera interfejsu graficznego. Wystarczy zaznaczyć jedną opcję:



Nazwę można zmienić lub pozostawić taką samą. Kreator zaproponuje nam utworzenie pliku CSS. Również powinniśmy się na to zgodzić:



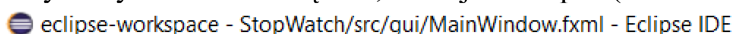
W ten oto sposób otrzymamy trzy pliki:

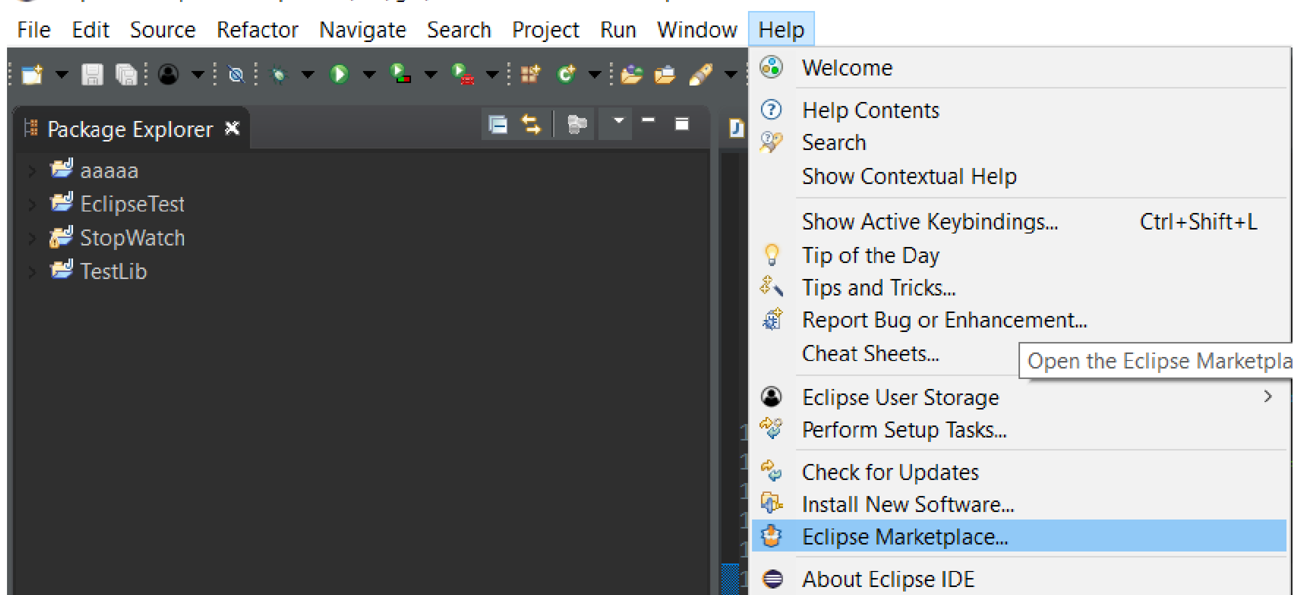
MainWindow.fxml – plik ten będzie zawierał wszystkie kontrolki, które będziemy dodawać do naszego okna (Java nazywa to sceną). Należy pamiętać, że tutaj również obowiązuje wykorzystywanie widoków.
mainwindow.css – plik będzie zawierał wszystkie ustawienia stylów każdej kontrolki
MainWindowController.java – plik klasy odpowiedzialnej za komunikację interfejs graficzny – wywoływanie funkcji. Klasa w tym pliku implementuje interfejs programistyczny Initializable, dzięki któremu można nadać pewne cechy i wykonać operacje w chwili tworzenia interfejsu graficznego (konkretnie naszego okna).

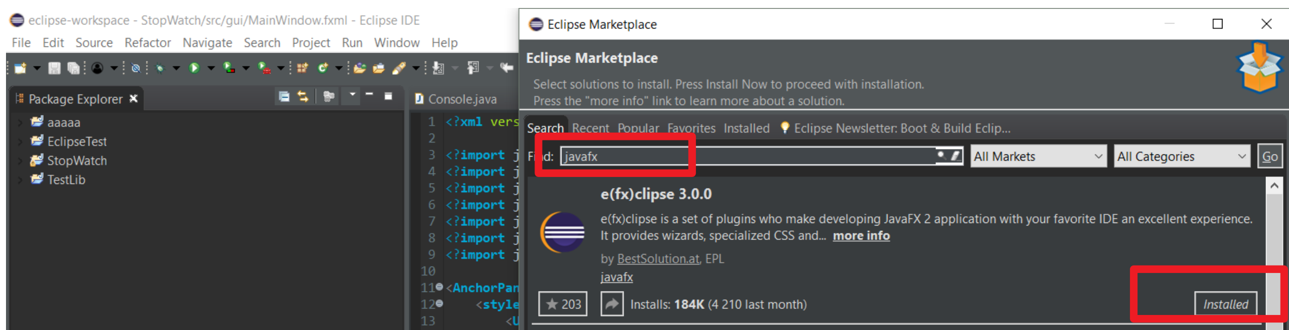
Rozpoczynamy pracę od dodania do programu odpowiednich kontrolek. W tym wypadku możemy dodać następujące elementy:
- listę wyświetlającą wszystkie czasy – ListView
- pole pozwalające na edycję opisu czasu – TextArea
- przycisk startu liczenia – Button
- przycisk zatrzymania liczenia czasu – Button
- przycisk chwilowego wstrzymania czasu (niedostępne w konsoli) - Button
- wybór reprezentacji czasu (niedostępne w konsoli) – ComboBox
- wyświetlanie aktualnego czasu (w odpowiedniej reprezentacji) – Text

Program można jeszcze wzbogacać o inne aspekty (jak dźwięk alarmu, odliczanie konkretnej wartości czasowej, zapis wyników do pliku itp.). Mając podstawową wersję aplikacji można następne jej elementy. W tym przykładzie chodzi jedynie o pokazanie możliwości JavaFX.

Niestety obecna wersja programu NetBeans (8.2) zawiera błąd, który nie podpowiada składni fxml. Błąd został zgłoszony na forum projektu, jednak od lipca ubiegłego roku nic się w tej materii nie zmieniło. Rozwiązaniem tej niedogodności może być:
- zainstalowanie najnowszej wersji rozwojowej; jednak wersja ta, chociaż podpowiada składnię, posiada nowy błąd – każdorazowo wyrzuca wyjątek tekstu podpowiedzi (problem zgłoszony, występuje nadal)
- zainstalować wcześniejszą wersję (8.1), która w pełni wyświetla podpowiedzi (ale jest sprzed ponad 3 lat)
- wykorzystać inne narzędzie, takie jak Eclipse (trzeba doinstalować dodatek):







W pole należy wpisać javafx (pierwszy prostokąt), a następnie kliknąć Install (drugi prostokąt, na zrzucie zainstalowane). Po zainstalowaniu i restarcie pliki fxml będą miały podpowiedzi składni.

Można też posłużyć się środowiskiem IntelliJ bądź równoważnym.

- napisać kod własnoręcznie i/lub kopiować fragmenty (nie działają podpowiedzi, ale działa sprawdzanie składni).

6.2 Właściwe tworzenie interfejsu.

Na początek należy wzbogacić naszą aplikację o klasę Application

```
public class StopWatch extends Application {

   public static void main(String[] args) {
      Console console = new Console();
      console.showMenu();
      launch(args);
   }

   @Override
   public void start(Stage primaryStage) throws Exception {
     //tutaj kod aplikacji
   }
}
```

Dodane fragmenty zostały pogrubione. Metoda start musi zostać dodana. Ponadto aby aplikacja się uruchomiła trzeba wywołać metodę (wbudowaną w klasę Application) launch, która niejawnie wywołuje start. Dodatkowo przesyłane są argumenty początkowe aplikacji (jeżeli takowe zostaną podane przez użytkownika; domyślnie przesyłany jest argument z nazwą wywoływanej aplikacji).

Metodę start można nadpisać dowolnymi instrukcjami, jednak najważniejszymi są następujące:

```
   @Override
   public void start(Stage primaryStage) throws Exception {
      Parent melement =
FXMLLoader.load(getClass().getResource("/gui/MainWindow.fxml"));
      primaryStage.setTitle("Nowa aplikacja");
      primaryStage.setScene(new Scene(melement));
      primaryStage.show();
   }
```

Wytłuszczone elementy są obligatoryjne. Pierwsza z nich ładuje element FXML ze źródła. W przypadku, gdy plik fxml przechowywany jest w osobnej paczce MUSI BYĆ poprzedzony znakiem slash (jak powyżej – przed nazwą paczki) inaczej interfejs NIE ZOSTANIE załadowany poprawnie.

Następnie do sceny (w rozumieniu tutaj – estrady, desek scenicznych) primaryStage dodajemy scenę (w rozumieniu tutaj – odgrywanej sceny sztuki, aktu) przygotowaną w pliku fxml.

Na koniec pokazujemy ją metodą show().

Opcjonalnym fragmentem jest ustawienie tytułu okna. Nazwa identyfikuje aplikację na pasku zadań czy menedżerze zadań.

Jeżeli zajdzie taka potrzeba, nasza aplikacja może posiadać własną ikonę identyfikującą ją np. na pasku zadań:

```
primaryStage.getIcons().add(new Image(StopWatch.class.getResourceAsStream("/indeks.png")));
```

Dzięki temu dodamy ikonę (z pliku png). Trzeba jednak pamiętać, że plik w chwili kompilacji musi znajdować się w katalogu źródeł (src). Ponieważ znajduje się on w głównej części tego katalogu w ścieżce posiada jedynie pojedynczy ukośnik (slash). Jeżeli byłby dodany do paczki (np. gui) to trzeba postąpić analogicznie jak w przypadku pliku fxml. Można też utworzyć ikonę z ze źródła w katalogu (przedrostek file:) lub ze zdalnych źródeł (od wersji 8 Java; przykład w kodzie).

Nasza aplikacja posiada obecnie jeden plik fxml. Oczywiście nic nie stoi na przeszkodzie by tych plików było więcej (np. odpowiedzialne za ładowanie plików, wyświetlanie komunikatów itp.) Struktura plików będzie przeważnie podobna:

```
<?xml version="1.0" encoding="UTF-8"?>

<?import java.lang.*?>
<?import java.net.*?>
<?import java.util.*?>
<?import javafx.scene.*?>
<?import javafx.collections.*?>
<?import javafx.scene.control.*?>
<?import javafx.scene.layout.*?>
<?import javafx.scene.control.ListView?>
<?import javafx.scene.text.*?>
<?import javafx.scene.input.*?>
<?import javafx.scene.control.cell.*?>

<AnchorPane id="AnchorPane" styleClass="mainFxmlClass" xmlns:fx="http://javafx.com/fxml/1"
fx:controller="gui.MainWindowController" >
   <stylesheets>
      <URL value="@mainwindow.css"/>
   </stylesheets>
   <VBox prefWidth="500" spacing="10" >
      <AnchorPane.topAnchor>10</AnchorPane.topAnchor>
      <AnchorPane.leftAnchor>15</AnchorPane.leftAnchor>
      <ListView fx:id="timeList" prefWidth="300" onMouseClicked="#listChange"  >
         <items>
            <FXCollections fx:factory="observableArrayList">
```

```
            <String fx:value="Test 1" />
          </FXCollections>
        </items>
      </ListView>
      <TextArea fx:id="timeDesc" prefHeight="100" onKeyReleased="#textAreaChange" />
    </VBox>
     <HBox prefHeight="30" spacing="10" alignment="BASELINE_LEFT" >
      <AnchorPane.leftAnchor>10</AnchorPane.leftAnchor>
      <AnchorPane.bottomAnchor>14</AnchorPane.bottomAnchor>
      <Button text="Start" styleClass="buttonStyle" onMouseClicked="#startTime" />
    </HBox>
</AnchorPane>
```

Powyżej został zaprezentowany jedynie fragment pełnego kodu. Najważniejsze elementy pliku:

- zawsze musi zaczynać się od linii
<?xml version="1.0" encoding="UTF-8"?>

Wynika to ze specyfikacji XML (wszystkie parsery XML jej poszukują, ponadto zawiera informację o kodowaniu znaków)

- dodatkowe importy paczek; podobnie jak w przypadku klas, pliki fxml również muszą posiadać „wiedzę" na temat dostępnych paczek i ich zawartości, jeżeli chcemy czegoś w nich użyć; podstawowe paczki i klasy zostały wytłuszczone, pozostałe zostały dodane w wyniku wykorzystania specyficznych klas (np. kolekcji danych)

- plik fxml musi znać nazwę pliku java z kodem kontrolującym gui. W tym wypadku jest to plik MainWindowController z paczki gui (wytłuszczony element). Może to być jednak zupełnie inaczej nazwany plik z dowolnej paczki (może to być nawet klasa główna aplikacji)

- wytłuszczony parametr fx:id odpowiada przede wszystkim za jednoznaczne zlokalizowanie kontrolki. Dzięki temu możemy przekazać go do kontrolera jako obiekt (w kodzie Java) i przez to się do niego odwołać w kodzie; dzięki temu możemy nadawać odpowiedni styl w pliku css (omówiony za chwilę); parametr ten ZAWSZE musi być unikatowy w obrębie pliku fxml

Pozostałe elementy, jak można wywnioskować, to nic innego jak nazwy klas kontrolek, które są zapisywane w notacji XML. Uwagę mogą zwrócić wytłuszczone zapisy AnchorPane. Jak widać są one opcjami elementu, które tutaj zapisane zostały jako osobne parametry obiektu. Wiele parametrów w plikach fxml posiada taki zapis (jak chociażby kolekcje, domyślne wartości itp.).

Plik CSS jedyne, czym różni się od CSS w stronach HTML, to zapisem właściwości. JavaFX wymaga, by właściwości zapisywać z przedrostkiem -fx- . Ponadto nie wszystkie właściwości działają (np. position, padding, margin itd.). Najlepiej jest sprawdzić w dokumentacji (materiał 11).

Ostatni pozostaje plik kontrolera. Jak zostało to wcześniej wspomniane, plik ten ma za zadanie obsługiwać graficzną postać aplikacji i łączyć ją z innymi klasami/obiektami aplikacji. Plik MainWindowController.java posiada wszelkie potrzebne komentarze. Tutaj przyjrzymy się tylko najważniejszym opcjom i właściwościom kodu:

- po pierwsze bardzo ważne jest skąd IDE Java pobierze informacje zarówno o kontrolkach jak i zdarzeniach je obsługujących. Poprawne będą te importy:

```
import javafx.scene.control.TextArea;
import javafx.scene.control.TextField;
import javafx.scene.input.KeyEvent;
import javafx.scene.input.MouseEvent;
```

niepoprawne np. te:

```
import java.awt.TextArea;
import java.awt.TextField;
import java.awt.event.MouseEvent;
import java.awt.event.KeyEvent;
```

Ani kontroli, ani zdarzenia nie są ze sobą kompatybilne pomiędzy rozwiązaniami. Dlatego jeżeli coś nie działa (a wedle przykładów powinno) należy sprawdzić to w pierwszej kolejności.

- komunikację pomiędzy obiektami z fxml a kodem Java może posiadać poprzez utworzenie tego typu zmiennej:

```
@FXML ListView timeList;
```

Adnotacja FXML mówi kompilatorowi, że kontrolka zdefiniowana jest w pliku fxml, natomiast nazwa timeList to nazwa fx:id z tamtego pliku (musi być ona dokładnie taka sama – inaczej połączenie nie zadziała).

W ten sam sposób zostały zdefiniowane pozostałe zmienne (konieczne w kodzie). Jeżeli dana kontrolka nie musi być użyta w kodzie można jej tutaj nie definiować.

- obsługa zdarzeń (np. kliknięcie przycisku) odbywa się w następujący sposób:

```
@FXML
   protected void stopTime(MouseEvent e) {
      mt.stopMeasure(((CellListView)
((HBox)timeList.getSelectionModel().getSelectedItem()).getChildren().get(0)).getText());
   }
```

w pliku fxml nazwa tej metody (poprzedzona krzyżykiem/krzakiem) pojawia się przy właściwości onMouseClicked . Chociaż przycisk nie jest przerzucony do kodu Java to może korzystać z metod, które są oznaczone adnotacją FXML. Ponieważ mamy do czynienia ze zdarzeniem myszy jak parametr przekazyany jest właśnie jej obiekt zdarzenia. Dodatkowo jest tutaj przykład odwołania się od metody obiektu mt, który został przekazany do kontrolera jako referencja z głównej klasy. Zdarzenie konretnie ma na celu zatrzymanie wybranego czasu. Aby tego dokonać trzeba posiadać jego nazwę. W tym wypadku trzeba odwołać się do szeregu referencji i przekonwertować obiekty (głównie Node) na odpowiednie klasy; tylko w ten sposób otrzyma się nazwę obiektu (getText). Trzeba mieć na uwadze, że z pozoru skompilowane odwołanie można rozbić na następujące elementy:

timeList.getSelectionModel().getSelectedItem() - pobiera obiekt z listy, który aktualnie jest na niej zaznaczony; ponieważ będzie zwrócony jako Object konwertujemy go na Hbox, stąd:

```
(HBox)timeList.getSelectionModel().getSelectedItem()
```

jednak potrzebujemy dostać się do kolejnej metody przekonwertowanego obiektu. Można w tej chwili utworzyć zmienną:

HBox hbotmp =(HBox)timeList.getSelectionModel().getSelectedItem();

jednak spowodowałoby to dodanie wielu niepotrzebnych zmiennych (nieużytecznych). Dlatego

(HBox)timeList.getSelectionModel().getSelectedItem()

otoczone zostaje nawiasem

((HBox)timeList.getSelectionModel().getSelectedItem())

co mówi kompilatorowi, że działanie z nawiasu ma zostać wykonane jako pierwsze. Wynik nawiasu będzie odpowiednim obiektem toteż pobieramy wszystkie dzieci-kontrolki, jakie on posiada:

(HBox)timeList.getSelectionModel().getSelectedItem()).getChildren()

jest to kolekcja,w tym wypadku zawierająca 2 kontrolki: pole tekstowe oraz przycisk (do usuwania nazwy z listy). Ponieważ znamy pozycję kontroli tekstu (pierwsza na liście), odwołujemy się do niej:

((HBox)timeList.getSelectionModel().getSelectedItem()).getChildren().get(0))

od razu obtaczamy polecenie nawiasami – chodzi nam o odwołanie się do tekstu elementu, który w ten sposób zdobędziemy. Kończąc łańcuch:

((CellListView)((HBox)timeList.getSelectionModel().
getSelectedItem()).getChildren().get(0)).getText()

otrzymujemy taką postać. Z elementu przekonwertowanego na CellListView (dziedziczy po TextField) możemy pobrać jego zawartość (nazwa czasu) i zatrzymać go.

INFORMACJA: W GUI bardzo często odwoływanie się do elementów i/lub pobieranie z nich informacji tak właśnie wygląda. W zależności od techniki albo stosuje się tego typu konwersje jedno liniowe lub odwołuje się do zmiennych (zapisuje konwersje pod odpowiednimi zmiennymi). Najlepiej jest stosować zasadę:

jednokrotne wykorzystanie – konwersja jednoliniowa
wielokrotne odwołania – zapis do zmiennej (efektywność).

Korzyść z zastosowania modelu MVC, a więc plików fxml, kontrolera i programu właściwego, widać dopiero w większych projektach, w przypadku pracy większej liczby, wyspecjalizowanych osób. Wtedy każdy może pracować nad swoim fragmentem aplikacji (np. menu) bez konieczności synchronizacji tej pracy np. z osobą zajmującą się komunikacją z bazą danych. W odpowiednim momencie tworzy się połączenia (zdarzenia) w aplikacji i jest ona w pełni gotowym produktem, w którym bez przeszkód można dokonywać zmian wizualnych lub mechaniki działania bez potrzeby modyfikacji elementów nie do zmiany.

6.3 Tworzenie i użytkowanie interfejsu HTML5 (WebView)

JavaFX daje jeszcze jedną możliwość budowania interfejsu graficznego użytkownika. Ponieważ posiada kontrolkę-obiekt WebView, możliwe staje się utworzenie całego interfejsu w języku HTML. Obecnie HTML5 jest jednym z najpopularniejszych (w zasadzie już najpopularniejszym) językiem projektowania dokumentów. W połączeniu z CSS i JavaScript (obie technologie są obecnie jego integralną częścią) możliwe staje się utworzenie aplikacji tzw. przeglądarkowych, czyli działających w jedynie w przeglądarkach WWW. Wykorzystując ServiceWorkers (element parsera JavaScript odpowiadający za dostarczenie wielowątkowości w JavaScript) można tworzyć aplikacje tzw. offline, czyli działające nawet w przypadku, gdy nasza aplikacja utraciła połączenie z serwerem.

Samo WebView wykorzystuje do renderowania stron silnik WebKit. To darmowy i otwartoźródłowy zestaw bibliotek odpowiadający za generowanie dokumentów HTML/XHTML. Wykorzystują go przeglądarki i projekty, takie jak Safari na OS X czy Blink w Chromium, Chrome, Opera.

Dzięki pierwotnej właściwości JavaScript (z której wzięła się jego nazwa), jaką jest możliwość komunikacji z obiektami aplikacji Java (a Java posiada obiekty i metody pozwalające na komunikację do JavaScript), możemy zbudować interfejs aplikacji znacznie szybciej, wygodniej i do tego w znacznie bardziej uniwersalny sposób. W odpowiedni sposób można bowiem utworzyć aplikacje HTML5 działającą po stronie serwera (np. przez przeglądarkę), która będzie miała dodatkowe możliwości i właściwości jeżeli będzie używana przez aplikację (np. zaawansowane zarządzanie stroną WWW, dzięki czemu panel administracyjny strony nie będzie dostępny dla standardowego użytkownika).

Plusy:
- uniwersalność – obecnie niemal każdy projektant graficzny zna HTML lub narzędzia, które znacznie ułatwiają projektowanie dokumentów WWW
- skalowalność – w różnych tego słowa znaczeniu. Po pierwsze, dzięki CSS, aplikacja może w prosty sposób posiadać interfejs dostosowujący się do rozdzielczości i wielkości ekranu. Ponadto aplikacja może być w prosty sposób rozwijana i posiadać dodatkowe wtyczki i/lub upiększenia, których nie doda się w JavaFX (np. edycja ramek kontrolek, nadawanie cieni, manipulowanie pozycją itp.). Aplikacja może też działać bez naszej aplikacji
- popularność – większość aplikacji powstaje jako HTML5. Znacznie łatwiej znaleźć osoby projektujące interfejsy w tej technologii niż JavaFX
- separacja – tak jak JavaFX, tak i HTML pozwala na pracę MVC. Kod JavaScript może być osobny, CSS może być osobny, HTML również. Do tego kojntroler Java może stanowić osobną klasę (mostki Java-JavaScript muszą być niezależnymi klasami). To powoduje, że projektowanie interfejsu w ten sposób jeszcze bardziej uniezależnia od siebie zespoły projektujące wygląd i funkcjonalność.

Minusy:
- powolność – w niedoświadczonych zespołach, w szczególności korzystających z rozwiązań takich jak Angular, jQuery itp., które ułatwiają pracę, może dojść do znacznego spowolnienia wykonywania operacji po stornie interfejsu. Chociaż pomocne, technologie te często wpływają negatywnie na efektywność aplikacji (a szczególnie młodzi programiści nie potrafią się bez nich obejść)
- podatność na ataki – w przypadku aplikacji na biurko ryzyko jest niemal zerowe; jeżeli jednak zespół zdecydowałby się na aplikację mieszaną (WWW/Java) trzeba mieć na uwadze zabezpieczenie każdego elementu krytycznego dla aplikacji. Jedno niepotrzebne połączenie może umożliwić złoczyńcom na włamanie się do kodu naszej aplikacji (tzw. code injection)

- niestabilność technologii – niestety, HTML5.x rozwija się bardzo szybko, to z kolei powoduje, że bardzo często różne jego elementy są dodawane, a niektóre usuwane (bądź są uznane za przestarzałe).

Aby skorzystać z HTML w swojej aplikacji trzeba odpowiednio przygotować kod Java. Będzie on podobny jak dla JavaFX:

```
WebEngine webEngine;

 @Override
   public void start(Stage primaryStage) throws Exception {

      WebView wv = new WebView();
      webEngine = wv.getEngine();
      webEngine.load(MyWebView.class.getResource("/webview/example.html").toString());
      primaryStage.setScene(new Scene(wv));
      primaryStage.show();
   }
```

Tworzymy tylko jeden element – WebView. Dodatkowo tworzymy zmienną referencyjną do silnika strony (WebEngine). Ładowanie strony następuje ze źródła scalonego z aplikacją (podane w paczce pliki html, css, graficzne i inne są bezpośrednio kompilowane do aplikacji). Dalsza część jest identyczna jak przy standardowej aplikacji JavaFX.

Tworzymy plik HTML:

```
<!DOCTYPE html>
<html>
   <head>
      <title>TODO supply a title</title>
      <meta charset="UTF-8">
      <meta name="viewport" content="width=device-width, initial-scale=1.0">
   </head>
   <body>
      <div>
         <select>
            <option value="0">sekundy [s]</option>
            <option value="1">mikrosekundy</option>
         </select>
      </div>
      <div>
      <ul id="timeList">
         <li><p contenteditable="true">default</p><button>X</button></li>
         <li class="addNewButton">Nowy czas...</li>
      </ul>
         <textarea></textarea>
      </div>
      <div><p id="output"></p></div>
      <div>
         <div class="buttons" id="startButton">Start</div>
         <div class="buttons" data-name="wstrzymaj">Wstrzymaj/wznów</div>
         <div class="buttons" data-name="zatrzymaj czas">Zatrzymaj</div>
```

```
      <div class="buttons" data-name="close">Zamknij program</div>
    </div>
  </body>
</html>
```

Posiada on prostą konstrukcję. Wyświetla pole typu ComboBox (tylko dwie opcje) oraz pole listy, która będzie edytowalna. Dodatkowo istnieje pole tekstowe (textarea). Na końcu istnieją 4 przyciski, którym dodano atrybuty data-*. To specjalne pola w HTML, które dodają własne atrybuty HTML, mogące być później czytane w JavaScript (i obiektach Java).

W pliku został umieszczony CSS nadający odpowiedni wygląd dodanym elementom:

```
<style>
      ul {
        list-style: none;
      }
      ul li {
        border: 1px solid black;
      }
      ul li p {
        margin: 0;
        padding: 0;
        margin-top: 5px;
        margin-left: 10px;
        margin-right: 10px;
        display: inline-block;
        width: 90%;
        font-size: 20px;
      }
      .addNewButton {
        background: black;
        color: white;
        text-transform:  uppercase;
        text-align: center;
        cursor: pointer;
        -webkit-user-select: none;
        user-select: none;
      }
      select {
        font-size: 20px;
        height: 30px;
        width: 50%;
      }
      .buttons {
        background: blue;
        color: white;
        font-weight: bold;
        text-transform: uppercase;
        display: inline-block;
        margin-right: 5px;
        padding: 25px;
        -webkit-user-select: none;
```

```
            user-select: none;
            cursor: pointer;
        }
        .buttons:hover {
            background: black;
        }
        textarea {
            width: 100%;
            height: 90px;
            font-size: 16px;
        }
    </style>
```

Jak widać jest to zwykły plik CSS.

Więcej wymaga omówienie JavaScript. Pierwsza z funkcji:

```
document.querySelector(".addNewButton").addEventListener("click", function (e) {
            var listopt = document.createElement('li');
            var listp = document.createElement('p');
            listp.innerHTML = 'nowy tekst';
            listp.setAttribute('contentEditable', true);
            var btn = document.createElement('button');
            btn.innerHTML = 'X';
            btn.addEventListener('click', function (e)
{ e.target.parentNode.parentNode.removeChild(e.target.parentNode);});
            listopt.appendChild(listp);
            listopt.appendChild(btn);
            document.querySelector('#timeList').insertBefore(listopt,e.target);
        });
```

odnajduje w dokumencie elementy, które posiadają klasę addNewButton, wybiera pierwszy z nich (w zasadzie jest tylko obecnie tylko jeden – dodający nowy element listy), i nadaje mu nowe zdarzenie, które aktywowane bezie kliknięciem. W tym momencie wykonają się następujące czynności:

- stworzony będzie element dokumentu li (element listy)
- stworzony będzie element p (paragraf)
- paragraf otrzyma nowy tekst z napisem ‘nowy tekst’
- dodany zostanie nowy atrybut, pozwalający na edycję zawartości paragrafy (ustawiony na true, czyli zostanie on włączony)
- utworzony zostanie element button (przycisk)
- dodany zostanie mu tekst (zawartość) ‘X’
- przyciskowi zostanie nadany nasłuch na zdarzenie kliknięcia, który będzie miał za zadanie:

a) znaleźć źródło zdarzenia (e.target)
b) przejść do rodzica tego elementu (parentNode), którym jest element listy (bo przycisk znajduje się w elemencie li)
c) znaleźć element nadrzędny (parentNode), którym jest cała lista
d) z tej listy usunąć dziecko, do którego należy przycisk (czy element, z którego nastąpiło kliknięcie)

- do elementu listy dodany zostanie paragraf
- do elementu listy dodany zostanie przycisk

- element listy zostanie dodany do listy, jednak zostanie dodany PRZED element wywołujący zdarzenie (bo nie jest zamierzeniem dodać zawartość po takim przycisku)

Kolejna ważna część kodu po stronie JavaScript:

```
 var btns = document.querySelectorAll('.buttons');

        for (var i=0;i<btns.length;i++) {
            if (btns[i].dataset.name)
                if (btns[i].dataset.name==='close')
                    btns[i].addEventListener('click', function (e) {windowObj.closeApp();});
                else if (btns[i].dataset.name==='zatrzymaj czas')
                    btns[i].addEventListener('click', function (e)
{document.winApp.testFunction('TEST');document.getElementById('output').innerHTML=e.target.
dataset.name; });
                else if (btns[i].dataset.name==='wstrzymaj')
                    btns[i].addEventListener('click', function (e)
{windowObj.testFunction(e.target.dataset.name);document.getElementById('output').innerHTML=
e.target.dataset.name; });
        }
```

- tworzymy zmienną-uchwyt do WSZYSTKICH elementów posiadających klasę 'buttons' (zamiast querySelector używamy querySelectorAll, tworzącą tablicę uchwytów do elementów HTML)
- następnie pętlą sprawdzamy, czy wskazany element tablicy posiada ustawioną zmienną name w kontenerze dataset (data-* po stronie HTML)
- jeżeli tak, sprawdzamy jego zawartość (można było zastosować switch→case)
- przyciski nie spełniają swojej znamionowej roli (poza przyciskiem zakończenia programu, który naprawdę zakończy program)
- przycisk posiadający wartość name 'zatrzymaj czas' uruchomi metodę z Java przekazana do JavaScript w obiekcie winApp (dodany do głównego dokumentu) o nazwie testFunction; przekazana zostanie do niej wartość 'TEST' poprzez parametr
- przycisk posiadający wartość name ' wstrzymaj' uruchomi metodę z Java przekazaną do JavaScript w obiekcie windowObj (została przekazana bezpośrednio do okna przeglądarki, dlatego nie wymaga odwołania się do obiektu document); jako parametr przekazana zostanie wartość zapisana w nazwie name elementu, który został wywołany (czyli 'wstrzymaj')
- przycisk z napisem 'close' wywoła metodę z obiektu Java, która wyłączy nasz program

Ostatnią metodą w JavaScript jest:

```
function showInfo(str) {
            document.getElementById('output').innerHTML=str;
            return "Odpowiedź z JS";
        }
```

Wypisuje ona w elemencie o id 'output' ciąg znakowy podany jako parametr (str). Ponadto zwraca wartość z odpowiednim komunikatem (String, chociaż dla JavaScript typ nie gra roli).

W kodzie głównym aplikacji dokonane zostało kilka zmian. Po pierwsze dodatkowe importy:

```
import netscape.javascript.JSObject;
import org.w3c.dom.Document;
import org.w3c.dom.Element;
```

```
import org.w3c.dom.events.EventListener;
import org.w3c.dom.events.EventTarget;
```

Pierwszy z nich, najstarszy i najważniejszy, umożliwia dwustronną komunikację pomiędzy Java i JavaScript.

Kolejne odpowiadają z dostęp do dokumentu, elementów zawartych w dokumencie, nadawaniu zdarzeń elementom po stronie HTML oraz pobieraniu celów zdarzeń (działa tak samo jak po stronie JavaScript). Należy zadbać by to były te paczki (mogą pojawić się inne, zupełnie niepotrzebne, psujące działanie całego programu).

W dalszej części dodane zostały dwie zmienne:

```
   Timer time;
   EventListener el;
```

Zmienne te zostały w przykładzie utworzone jako globalne, mogą jednak być lokalne (można je utworzyć w miejscu ich definicji, dzięki powyższemu zapisowi program ma po prostu do nich dostęp z każdego poziomu).

W kodzie repozytorium metoda start() rozpoczyna się od następujących linii:

```
el = new EventListener() {

            @Override
            public void handleEvent(org.w3c.dom.events.Event evt) {
                System.out.println("Wywołane z JavaScript");
            }
        };
```

Nie muszą one występować dokładnie (mogły zostać utworzone w elemencie poniżej), jednak to nic nie wnosi. Najważniejsze jest, że tworzymy tutaj nowy nasłuch zdarzenia, który jeżeli zostanie wywołany, wyświetli nam stosowny komunikat z konsoli. Jak można zauważyć, mamy tutaj do czynienia z pełną implementacją zdarzeń JavaScript (i na odwrót).

Następnie w metodzie start() MUSI znaleźć się następujący nasłuch wydarzeń:

```
webEngine.getLoadWorker().stateProperty().addListener((ObservableValue<? extends State>
observable, State oldValue, State newValue) -> {

});
```

W tym wypadku tworzymy zdarzenie, które będzie nasłuchiwało KAŻDEJ zmiany stanu wczytanego dokumentu do WebView. Anonimowa metoda (lambda) musi przechwycić trzy parametry
- polimorficzną postać interfejsu ObservableValue, ktoy będzie przechowywał wartość obiektu, którego zmian nasłuchuje
- poprzednią wartość stanu, w której obiekt się znajdował
- nową wartość stanu, w której obiekt się znajduje

Oczywiście można implementować obsługę wszystkich zmiennych, nas jednak będzie interesowała nowa wartość (pozostałe nam się teraz nie przydadzą). Interesuje nas dokładnie ta wartość:

```
if (newValue==State.SUCCEEDED) {

}
```

czyli udane wczytanie dokumentu.

Po pierwsze zmienimy nazwę okna (należy zauważyć, że jeżeli dokument by się nie wczytał, okno nie miałoby tytułu:

```
primaryStage.setTitle("Aplikacja HTML");
```

Następnie te dwie linie

```
JSObject win = (JSObject) webEngine.getDocument();
        win.setMember("winApp", new MyBridge());
```

odpowiednio tworzą dowiązanie do obiektu JSObject poprzez pobranie załadowanego dokumentu, a później dołączają do niego nową zmienną (wartość) winApp, która będzie obiektem klasy MyBridge (stworzony wcześniej w kodzie). Możliwe jest też stworzenie zmiennej globalnej tego typu obiektu celem dokonywania w nich zmian (np. przekazanych wartości itp.). Wtedy podaje się nazwę zmiennej, pod którą kryje się utworzony obiekt.

Następne dwie linie:

```
win = (JSObject) webEngine.executeScript("window");
        win.setMember("windowObj", new CommunicationBridge());
```

działają analogicznie do wcześniejszych. Różnicą jest, iż zamiast pobierać dokument wywołujemy polecenie skryptu. Ponieważ poleceniem tym jest wartość z JavaScript odnosząca się do drzewka BOM (okna przeglądarki), JSObject zwraca do zmiennej win uchwyt do tej zmiennej. Wykonując polecenie setMember, wykonuje się ono na przechwyconej zmiennej. Dlatego później, w JavaScript, zmienną windowObj można wykonywać bez jawnego podawania obiektu nadrzędnego (jest częścią okna).

Linie:

```
Document d = webEngine.getDocument();
        Element e = d.getElementById("startButton");
        ((EventTarget)e).addEventListener("click", el, true);
```

kolejno:
- tworzą uchwyt do załadowanego dokumentu w WebView
- z dokumentu wyciągają uchwyt do elementu z identyfikatorem (atrybut id) o nazwie „startButton"
- dodają do niego nasłuch zdarzenia na kliknięcie (‘click’); jeżeli nastąpi ma się wykonać zdarzenie el (utworzone na samym początku metody start()). Ostatnia wartość określa sposób przechwycenia zdarzenia (poprzez tzw. bąbelkowanie lub poprzez przechwycenie, więcej w materiale 28).

Tak oto z Java możemy nadawać nowe zdarzenia elementom HTML (nawet nadpisywać czy usuwać – metodą removeEventListener).

Ostatni fragment kodu jest analogiczny do tego znanego z JavaFX:

```
time = new Timer();
        time.scheduleAtFixedRate(new TimerTask() {
            @Override
            public void run() {
                Platform.runLater(()->{
                    executeCommand();
                });
            }
        }, 1000, 1000);
```
Co sekundę (1000 milisekund) wywoływana będzie metoda executeCommand(). Musiała ona zostać wpisana w metodę runLater(), w przeciwnym wypadku otrzymalibyśmy wyjątek, że próbujemy komunikować się pomiędzy zawartością kontrolki FX z niewłaściwego wątku (Timer odpala własny, niezależny wątek). Obiekt Platform jest pomocniczą klasą odpowiadającą za stan aplikacji głównej. Jego metoda runLater uruchamia wątek, który odpalany jest w tej samej puli co GUI (dlatego wszystko zadziała jak należy).

Metoda executeCommand wygląda następująco:

```
public void executeCommand() {
      String odp = (String) webEngine.executeScript("showInfo('Informacja z Java!')");
      System.out.println(odp);
   }
```

Pierwsza linia uruchamia metodę JavaScript, do której przekazujemy parametr Informacja z Java! Dodatkowo to co ona zwraca (tekst) przekazujemy do zmiennej odp, którą wyświetlamy z konsoli.

Przykład, chociaż nie w pełni działający (jak ten z JavaFX) pokazuje wszystkie aspekty niezbędne do tworzenia aplikacji z GUI HTML5, wraz z obustronną komunikacją.

7. Podsumowanie

W powyższym materiale przedstawione zostały wszystkie NOWE/PREFEROWANE możliwości tworzenia interfejsu w Java. Oczywiście można używać AWT lub Swing, jednak są to elementy uważane za przestarzałe. Jedną z ciekawszych opcji jest tworzenie interfejsu w pełni HTML. Posiada tę przewagę nad pozostałymi opcjami, że jest najbardziej dojrzały, wykorzystuje mechanizm połączenia z Java (dostępny od połowy lat 90) oraz rozwija się niezależnie od innych technologii. Jego wadą będzie na pewno mniejsza efektywność, zwłaszcza w przypadku wykorzystywania przez mniej doświadczone osoby.

Materiały:

1) https://pl.wikipedia.org/wiki/JavaFX
2) https://wheelercode.wordpress.com/javafx-css-properties-selectors-list/
3) https://docs.oracle.com/javafx/2/layout/builtin_layouts.htm#CHDGHCDG
4) https://docs.oracle.com/javafx/2/ui_controls/overview.htm
5) https://docs.oracle.com/javafx/2/ui_controls/radio-button.htm
6) https://www.programcreek.com/java-api-examples/?api=javafx.scene.layout.Background
7) https://docs.oracle.com/javafx/2/ui_controls/text-field.htm
8) http://www.massapi.com/method/javafx/scene/control/Button.onMouseClickedProperty.html
9) https://docs.oracle.com/javafx/2/events/handlers.htm
10) https://www.tutorialspoint.com/javafx/javafx_event_handling.htm
11) https://docs.oracle.com/javase/8/javafx/api/javafx/scene/doc-files/cssref.html
12) https://www.callicoder.com/javafx-css-tutorial/

13) https://stackoverflow.com/questions/23726899/change-console-input-encoding-in-netbeans-8-0
14) https://stackoverflow.com/questions/2415597/java-how-to-detect-and-change-encoding-of-system-console/2415629
15) https://stackoverflow.com/questions/4964640/reading-inputstream-as-utf-8
16) https://stackoverflow.com/questions/12944377/how-to-convert-byte-to-byte-and-the-other-way-around
17) https://docs.oracle.com/javase/7/docs/api/java/io/OutputStreamWriter.html
18) https://stackoverflow.com/questions/23709515/netbeans-java-console-encoding-utf-8-and-umlauts
19) https://stackoverflow.com/questions/2979383/java-clear-the-console
20) https://stackoverflow.com/questions/5585919/creating-unicode-character-from-its-number
21) https://gist.github.com/jewelsea/3062859
22) https://stackoverflow.com/questions/10121991/javafx-application-icon
23) https://stackoverflow.com/questions/4044726/how-to-set-a-timer-in-java
24) https://stackoverflow.com/questions/18244943/javafx-fxml-how-do-i-set-the-default-selected-item-in-a-choicebox-in-fxml
25) https://stackoverflow.com/questions/15819242/how-to-make-a-button-appear-to-have-been-clicked-or-selected-javafx2
26) https://www.tutorialspoint.com/javafx/javafx_css.htm
27) https://www.callicoder.com/javafx-fxml-form-gui-tutorial/
28) https://javascript.info/bubbling-and-capturing
29) https://blogs.oracle.com/java/javafx-webview-overview
30) https://stackoverflow.com/questions/35985601/calling-a-java-method-from-a-javafx-webview
31) https://stackoverflow.com/questions/39725360/access-return-value-from-a-javascript-in-javafx
32) https://stackoverflow.com/questions/48955193/javafx-calling-javascript-to-excute-java-function-in-webview-not-working
33) https://www.programcreek.com/java-api-examples/?class=netscape.javascript.JSObject&method=setMember
34) https://stackoverflow.com/questions/26916640/javafx-not-on-fx-application-thread-when-using-timer
35) https://stackoverflow.com/questions/48721849/javafx-webengine-executing-javascript-sequentially-to-do-actions-on-the-website
36) https://www.programcreek.com/java-api-examples/?api=javafx.concurrent.Worker.State